1 Variational Tools for Signal Similarity Analysis

Authors: Selim Esedoglu and Kevin Vixie.

Affiliation:
Esedoglu: Department of Mathematics, University of Michigan
Vixie: Los Alamos National Laboratory.

Contact Information:
Esedoglu: University of Michigan
Department of Mathematics
2074 East Hall
530 Church St.
Ann Arbor, MI 48109
Phone: (734)-936-9936
email: esedoglu@umich.edu

Vixie: Mathematical Modeling and Analysis, T-7
Mail Stop, B284
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87545
Phone: (505)-665-9887 email: vixie@lanl.gov

Description

This collection of eight routines, and their associated subroutines, implement a number of metrics for quantifying the differences between signals from experiments and simulations.

Our approach has been informed by the intuitive process by which experts judge the similarity and differences between two given signals (typically, one signal from experiments, and the other from simulation).

An important criterion that experts have been observed to judge the signal similarity by is closeness of the *envelopes* of the two signals. This has motivated us to develop a mathematical, multi-scale notion of the envelope of a given time series signal, using an approach based on the calculus of variations. Our models allow the user to specify how tightly (i.e. at how fine a scale) the envelope should follow the underlying signal, as well as how hard the constraint of being an envelope (i.e. always remaining above or below the graph) is to be enforced.

Furthermore, the expert opinion seems to suggest that if two signals are slightly out of phase (even if in a non-uniform way), they may be still considered to be in excellent agreement if their overall energy and peaks and nadirs are in close correspondence. In order to compare signals modulo such apparently insignificant variations in phase, we developed an algorithm that can locally stretch or compress (i.e. deform) one signal to match as closely as possible the other. Naturally, a penalty is imposed on the extent of the deformation required for the match, which then serves to quantify the similarity.

Another important way to suppress insignificant signal details is through the classical technique known as Tikhonov regularization. In this variational approach, a smoother form of the signal is obtained by attenuating higher frequencies. This way, larger scale features of the signals can be exposed and compared, instead of getting lost in the much finer and unreliable attributes. We have implemented Tikhonov regularization as well, and make use of it in connection with the other tools developed as mentioned above.

A more specific description of the individual routines follows:

`tik = tik_f(fin,lambda):`

This routine computes the Tikhonov regularized version of the input signal `fin`. The regularized signal is returned as output. The parameter lambda determines the scale at which the input signal is regularized: The larger the parameter, the less regularized (and hence finer) the output signal. As the parameter lambda is decreased, the regularized signal is allowed to deviate more and more from the given one for the sake of increased regularity.

`[enu,enl] = env_ev(nsteps,dt,in_enu,in_enl,fin,lambda,mu)`

Given an input signal represented by `fin`, this routine computes an upper envelope function, denoted `enu`, and a lower envelope function, denoted `enl`, that encapsulate the graph of `fin`. It implements a variational model for finding these envelope functions, and solves the nonlinear optimization problem involved iteratively, using a number of time steps specified by `nsteps`. The user can set the scale (i.e. the detail level, or how closely the envelope functions follow ups and downs of `fin`) by tuning the parameter `lambda`. The user can also set how strictly the two "constraints" `enu`$\geq$`fin` and `enl`$\leq$`fin` are to be enforced by tuning the parameter `mu`.

`[u,E] = match(nt,dt,uin,f,g,fid)`

This routine is used for warping a one dimensional signal `f` onto another given by `g`, so that the two match as well as possible while minimizing a warping energy. It is based on a variational model that involves minimizing a cost function iteratively. Therefore, the routine requires the number of iterations `nt` to take and the step size `dt` used in each iteration, as well as an initial guess `uin` for the deformation, to be specified. Moreover, the user is able to tune how aggressive the deformation should take place by tuning the parameter `fid`. The routine returns not only the optimal warping function u found, but also its corresponding optimal cost E.

`y = fint1d(x,f)`

This routine carries out linear interpolation in one dimension. The input function `f` is assumed to be defined on a uniform grid on the unit interval $[0, 1]$; the first grid point is assumed to be at 0, and the last one at 1. The input vector `x` contains the coordinates of points at which `f` is to be interpolated.

`d = warp_dist(nt,dt,f,g,fid)`

This routine calls on `match.m` twice in order to compute a qualitative measure of difference between the two input signals `f` and `g` up to deformations of one onto the other (so as to compensate for small phase differences and the like). First, `f` is warped onto `g`, and the corresponding cost recorded, and then `g` is mapped onto `f`. The value `d` returned by the routine is the sum of the two deformations costs.

`[env_measures] = env_meas(e)`

This routine takes as its input the upper and lower envelope of a one dimensional signal and computes several measures from them that characterize certain aspects of the signal. Considering the envelope as the region between the graphs of the upper and lower envelope functions, it computes:

1. The vertical centroid of the envelope region,
2. The widest gap between the upper and lower envelopes,
3. Location where the maximum gap between upper and lower envelopes occurs,
4. The vertical width of the envelope,
5. Location where the upper envelope is largest,
6. Location where the lower envelope is smallest,
7. Maximum of the upper envelope,
8. Minimum of the lower envelope,
9. Area of the envelope region (i.e. area between the graphs of the upper and lower envelope functions).

`[symdiff] = env_symdiff(e1,e2)`

This routine is used to compare how close the envelops obtained from two different one dimensional signals are. The envelope of a function is thought of as the region between the its upper and lower envelope functions (which can be obtained from the routine `env_ev.m`). This routine compares the envelopes by finding the area of symmetric difference (i.e. the set of points defined as being contained in only one of) the two regions. This measure is then non-dimensionalized by dividing it by the sum of the areas of the two regions.

Mathematical Principles

`tik = tik_f(fin,lambda):`

Tikhonov regularization is a classical variational model for smoothing a given signal $f(x)$. It exhibits the regularized version of $f(x)$ as the solution of the following optimization problem:

$$\min_u \int |\nabla u|^2 + \lambda\big(u(x) - f(x)\big)^2\, dx$$

This is a strictly convex variational problem with a unique minimizer. Furthermore, the Euler-Lagrange equation (i.e. the optimality condition) satisfied by the minimizer is the following *linear* PDE:

$$\Delta u + 2\lambda\big(u - f\big) = 0.$$

This simple, linear elliptic PDE can be easily solved numerically; the routine $tik_f.m$ utilizes the three point formula to approximate the Laplacian operator on the uniform grid:

$$\Delta u(x_j) \approx \frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2}$$

where h stands for the uniform grid size (the domain of the signal is assumed to be $[0, 1]$). The discrete form of the PDE is therefore

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + 2\lambda(f_j - u_j) = 0.$$

The resulting linear system of equations is then solved very efficiently using the discrete Fourier transform. This is accomplished as follows: In the frequency domain, the three point formula for the Laplacian becomes the multiplication operator:

$$\hat{u}_j \to (w^{j-1} - 2 + w^{1-j})\hat{u}_j$$

which is very easy to invert in this basis, since it is *diagonalized*. Here, w denotes the n-th root of unity $\exp(2\pi i/n)$, where n is the length of the input signal. The discretized PDE becomes:

$$\big(w^{1-j} - 2 + w^{j-1} - 2\lambda h^2\big)\hat{u}_j = -2h^2\lambda\hat{f}_j.$$

After solving for $\hat{u}_j$ above, an inverse DFT produces the desired result.

`[enu,enl] = env_ev(nsteps,dt,in_enu,in_enl,fin,lambda,mu)`

We propose a mathematical notion for the "envelope" of a given function $f(x)$, using a variational formulation. Any reasonable definition of envelope must necessarily allow for "scale selection"; in other words, it should come with a parameter that determines how closely the envelope should follow oscillations in the graph of $f(x)$. This depends on the size of features that are relevant in the graph of $f(x)$.

We describe the envelope of $f(x)$ using two functions: The upper envelope, $u(x)$ and the lower envelope $v(x)$ The graph of the upper envelope function should remain as much as possible above the graph of the given function $f(x)$, whereas the graph of the lower envelope should remain below it; how rigidly these conditions are enforced needs to be user selectable so that large spikes that may be due to noise can be ignored if necessary. In addition, the graphs of $u(x)$ and $v(x)$ should follow the oscillations in the graph of $f(x)$ by staying close to it. Once again, how closely u and v should follow f must also be user tunable.

Our model for finding the upper envelope u consists in solving the following optimization problem:

$$\min_u \int |\nabla u|^2 + \lambda u + \mu\Big((f-u)_+\Big)^2\, dx$$

The two constants λ and μ are to be chosen positive, with μ typically much larger than λ. In this energy, the first term penalizes large gradients and rapid oscillations in u, so that the envelope function turns out to be a simpler representation of the signal, with all its tiny oscillations suppressed if desired. The second term in the energy encourages u to be as small as possible; this is the term that attracts the graph of u towards that of f. The larger the parameter λ, the closer the graph of u follows that of f, leading to a more detailed envelope (i.e. one that captures finer scale features). The third term is acts as a barrier, turning on only at points where u falls below f, and counteracting to keep the graph of u above that of f.

The Euler-Lagrange equations for this model lead to the following gradient descent PDE:

$$u_t = 2\Delta u - \lambda + 2\mu(f-u)_+ H(f-u)$$

where $H(\xi) : \mathbb{R} \to \mathbb{R}$ denotes the Heaviside function, i.e. $H(\xi) = 1$ if $\xi > 0$ and $H(\xi) = 0$ otherwise. The following semi-implicit time discretization is used to solve this PDE:

$$\frac{u^{n+1}-u^n}{\delta t} = 2\Delta u^{n+1} - \lambda + 2\mu(f-u^n)_+ H(f-u^n)$$

where δt is the time step size and u^n denotes the solution at time $t = n\delta t$. This equation can now be discretized also in space and the resulting linear system solved at every time step using once again the discrete Fourier transform, in much the same way that was explained in the description of the routine `tik_f.m`.

To find the lower envelope v, we propose solving the following optimization problem:

$$\min_v \int |\nabla v|^2 - \lambda v + \mu\Big((f-v)_-\Big)^2\, dx$$

the gradient descent for which gives:

$$v_t = 2\Delta v + \lambda - 2\mu(f-v)_- H(f-v)$$

Its interpretation and numerical treatment is completely analogous to the ones for the upper envelope u described above.

```
[u,E] = match(nt,dt,uin,f,g,fid)
```

This routine implements a variational model for warping a one dimensional function f onto another g. The two signals are assumed to be defined on the unit interval $[0,1]$. Then, the mapping sought is a deformation of $[0,1]$ into itself, i.e. a map $u : [0,1] \to [0,1]$. Using this deformation, new images are generated from f by the composition $f(u(x))$; these are warped versions of f. The model looks for the map $u : [0,1] \to [0,1]$ that best matches $f(u(x))$ to the other given signal $g(x)$ by minimizing the following cost function:

$$\int_0^1 |\nabla u|^2 + \lambda\Big(f(u(x)) - g(x)\Big)^2\, dx.$$

The first term in the energy imposes a smoothness penalty on the deformation function u so that the mapping is not too unruly (and in particular, takes nearby points to nearby points). The relative importance of this term with respect to the "fitting" term can be tuned via the parameter λ.

The Euler-Lagrange equations that minimizers of the deformation model satisfies leads to the following gradient descent PDE that can be used to seek them out:

$$u_t = \Delta u - \lambda\big(f(u) - g\big)f'(u).$$

The time discretization used in the numerical solution of this PDE is:

$$\frac{u^{n+1}-u^n}{\delta t} = \Delta u^{n+1} - \lambda\big(f(u^n) - g\big)f'(u^n)$$

where u^n denotes the approximate solution at time $t = n\delta t$. As with the previous models, the resulting linear system at every time step can be efficiently solved via the discrete Fourier transform. An important numerical point is that the evaluations $f(u^n)$ and $f'(u^n)$ require interpolation (since the functions f and f' are defined on the grid, but $u(x)$ need not correspond to a grid point). Linear interpolation, provided by the routine `fint1d.m` was used for this purpose.

Physical and Engineering Principles

Our metrics and measures were constructed not so much as any sort of final answer (in their raw form), but rather to provide representations of the differences between signals which would be useful for building a good metric. In particular, we believe that the output of our measure and distance functions will provide good coordinates in which to build custom, weighted metrics. Having components or coordinates capturing the right differences and features is important, even if we do not believe any one of the coordinates will do the trick.

Our approach was inspired by listening to the subject area experts discuss how they judged closeness of simulated and experimental data. What emerged from those discussions was that the features such as peak amplitudes, when those peak amplitudes occurred, decay envelopes, and vertical shifts of those envelopes were all used in the evaluation of how well two signals matched or did not match. Additional features included energy content and decay, and frequency content.

Envelopes and Tikhonov regularized versions of the signals, each computed at a small set of scales, were chosen because they seem to encapsulate a good fraction of what was being used to make expert comparisons. In particular, they use or measure:

(1) decay

(2) max peak height

(3) max peak location

(4) energy

(5) frequency content

A metric constructed using the multiscale measure/distance vector we compute should therefore be able to "see" these important features.

1.1 Usage

In this subsection we give, for each function and driving routine, a brief synopsis of usage and things to know about parameter choices, instabilities to avoid, etc. The synopsis of each is taken from the documented `.m` files.

The functions:

1. `test1116.m` is a driving program generating the metric/distance metrics form a collection of experiment/simulation pairs.

2. `metrics_SE_KRV.m` is the program that generates the metric/distance vectors given a single pair of signals. The choice of the tuning parameters were arrived at empirically. For a discussion of choices for λ, μ, dt, and *nsteps* see the usage notes for `env_ev` and `Tik_f`.

3. `warp_dist.m` is a function which computes the warping distance between to input signals

4. `env_symdif.m` is a simple function which computes the area of the symmetric difference between two input envelopes. This is then divided by the sum of the areas of the two envelopes, yielding a dimensionless number between 0 and 1. The use of this function is completely straightforward.

5. `env_meas.m` is a function with computes 9 simple measures from a single envelope. The use of this function is completely straightforward.

6. `fint1d.m` is a function which computes interpolations

7. `match.m` is a function which finds the optimal warp of one function into another. Since this is not symmetric in the input functions, `warp_dist.m` calls this function twice to symmetrize the resulting warping distance.

8. `env_ev.m` computes the envelope of an input signal. There are tuning or scale parameters controlling how smooth and how strict the envelope is.

9. `tik_f.m` computes the Tikhonov regularization of an input signal, The tuning or scale parameter controls how smooth vs. how faithful the output is.

10. `expert_metric_comp.m` is a currently *untested* function which compares any of the distances we compute to an expert supplied scoring (in the form of a distance). We include this function because it was carefully written (not too many bugs probably) and, more importantly because we think that the idea it contains is valuable.

test1116.m

```
inputs:

 fname = name of the mat file
          including the .mat extension

 Test  = flag choosing tuning or testing (validation).
         Set true causes it to be run on the
         non-tuning data.
outputs:

  M = a cell array of results for
      the comparisons between experimental
      and simulation metric output.  Our
      metric code does not have time of Arrival
      code in it, so we assume the input is
      properly truncated (or at least all have
      the same truncation).
      Each cell M{i,j}() is a 49xk matrix, each
      column of which contains the differences
      of measures between the experiment and simulation
      or the various distances.  To be precise, the
      first 36 elements are differences of measures
      -- simulation minus experiment -- the next
      12 are distances, two sets of fractional
       symmetric differences and one set of warping
      distances.The number k is controlled by the
      number of experiments  and equals the number
      of columns in Ce{i,j}().

M_label = 49 x 1 cell array of strings
           containing more complete descriptions
           what the measures or distances are.

notes:

   1) the first 36 entries of each column vector
```

```
      in M can be negative, the last 13 are always
      positive
   2) This is really just a calling program.  There
      is the test/tuning parameter that can be set.
      In particular

               tt_select = (some number in [0,1])

       controls the relative size  tuning/testing sets
```

metrics_SE_KRV.m

```
[labels,measures]=metrics_SE_KRV(ref)
[labels,measures]=Metrics_SE_KRV(ref,data)


synopsis: computes several measures and metrics on input
          column vectors.

input:

ref  =  is a reference vector of doubles.  For the Threaded
        assembly data, it represents a time signal output
        from a strain gauge or an accelerometer.  It can
        be any vector of values, but some of the measures
        may not make sense out of context.

data =  is a data vector of boubles for comparison with
        ref.  It need not be the same length as ref, but
        the both the data and ref are assumed pre-truncated.

output:

labels   =      an N by 1 cell array of strings that identify
                the corresponding measures or metrics (see below).
measures    =   an N by 3 vector of doubles.  Each row corresponds
                to either measures on the input data OR a single
                metric distance between the two inputs.

parameters:

  N = 49 = is the number of measures or metrics returned
  T        if T = 1, do Tikonov reg preprocessing. If T = 0 skip it.
           If it is done, a scale of 5000000 seems to work well
           for normalized signals and time in ticks.
L_gen =    if = 1 then labels are generic otherwise customized
           labels are output.

 notes:

    1) very briefly, we compute envelopes of both curves (if there are two)
       or the single curve and then generate 9 measures on each envelope.
       If there are two curves, then several distances between the curves
```

```
       are computed using the envelopes of each curves computed at several
       scales, envelopes of the spectrum (abs(fft(signal))) of each curve and
       the warping distance is computed between Tikonov regularized versions
       of the two curves.

    2) The scales used (see tik_scale, l_scale, m_scale, t_step below) have
       been arrived at empirically.  some more thought might be inversted
       here later, but for now this seems to cover a good range and work
       well of the data with ticks for time and the frequency content that
       we have seen in these simulations and experiments.
```

warp_dist.m

```
[d] = warp_dist(nt,dt,f,g,fid)

synopsis:

 This function computes the warping distance between
 the two input images f and g, by deforming the
 underlying space. It does this by finding the optimal
 deformation from f to g, and from g to f, by calling
 the routine match.m twice.

input:

  nt =  Number of time steps to take in finding the
        the optimal warping.
  dt =  Time step size in iteratively finding the
        optimal warping.
   f =  Input image whose distance to g is to be found.
   g =  Input image whose distance to f is to be found.
 fid =  Strength of fidelity term in warping model.

output:

   d = The warping distance. Although symmetric, it
       probably is not a true "metric".

notes:

   1) The input images f and g are assumed to be row
      vectors of equal length.
   2) There is no error check!
```

env_symdif.m

```
[symdiff] = env_symdif(e1,e2)

synopsis:   computes a dimensionless measure of symmetric
              difference between two input envelopes.

input:

  e1,e2  =  the two envelopes which are each assumed to be
            a 2 by n matrix where the first row e(1,:) is
            the upper envelope and the second row e(2,:) is
            the lower envelope.

output:

 symdiff =  the area of the symmetric differece of the
            envelopes divided by the sum of the areas of
            the envelopes.

notes:

     1) The input y values are assumed to be from points uniformly
        spaced in x with a dx = 1.  Simply multiply the answer
        returned by the correct dx if it is not = 1.

     2) if the two envelops are not of equal length, then one
        is truncated
```

env_meas.m

```
[env_measures]= env_meas(e)


synopsis: compute several measures from an envelope


input:
       e = 2 by n matrix, e(1,:) assumed the upper envelope
           e(2,:) assumed to be the lower envelope

output:

 env_measures = 9 dimensional row vector containing the
                computed measures.  They are, in order
                of appearance in env_measures,

                y_centroid = vertical centroid of envelope
                max_vert   = max (e(1,:)-e(2,:))
                max_vert_x = argmax (e(1,:)-e(2,:))
                vert_width = max(e(1,:)) - min(e(2,:))
                x_max      = argmax e(1,:)
                x_min      = argmin e(2,:)
```

```
                y_max      = max e(1,:)
                y_min      = max e(2,:)
                area       = area enclosed by envelope

   notes:

           1) The input y values are assumed to be from points
              uniformly spaced in x with a dx = 1.  Simply multiply
              the answer returned by the correct dx if it is not = 1.
```

fint1d.m

```
[y] = fint1d(x,f)

synopsis:

   This routine carries out linear interpolation in one dimension.
   The input function f is assumed to be defined on a uniform
   grid on the unit interval [0,1]; the first grid point is
   assumed to be at 0, and the last one at 1. The input vector
   x contains the coordinates of points at which f is to be
   interpolated.

input:
      x = Location of points where value of f(x) is desired.
          Assumed to be a row vector, same size as f.
      f = Values of the function at the the uniform grid points.
          Assumed to be a row vector, same size as u.

output:
      y = Interpolated values of f at the points in x.
          Row vector of same length as x and f.
```

match.m

```
[u,E] = match(nt,dt,uin,f,g,fid)

synopsis:

      Routine used for warping the one dimensional signal f
      onto g by deforming the interval on which they are defined.

 input:
      nt  =  Number of time steps to take.
      dt  =  Time step size.
      uin =  Initial guess for the deformation function.
```

```
      f   =  Signal to be deformed onto g.
      g   =  Signal onto which f is to be deformed.
      fid =  Fidelity constant appearing in the deformation model.

output:
      u =  Optimal deformation computed by the model.
      E =  Energy (i.e. cost) of the optimal deformation.
```

env_ev.m

```
[enu,enl] = env_ev(nsteps,dt,in_enu,in_enl,fin,lambda,mu)

synopsis: find an envelope of a 1-d signal or function
          (env_ev = envelope evolver)

input:

   nsteps  =  number of steps
       dt  =  time step size
   in_enu  =  initial upper envelope
   in_enl  =  initial lower envelope
      fin  =  function for which the envelope
              is desired.
    lambda =  smoothing parameter (the balance
              between the regularization and the
              data fidelity part for curve "outside"
              curve)
       mu  =  smoothing parameter (the balance
              between the regularization and the
              data fidelity part for curve "inside"
              curve)

output:

     enu =  upper envelope
     enl =  lower envelope

notes:

  1) we assume data entered is a row vector
  2) we don't do any error checking
  3) there are other versions that are curently
     commented
     out below
  4) We found that for signals normalized to take values in [-1, 1],
     time in ticks and the types of oscillations that seem typical
     in the empirical signals the following choices for {dt,lambda,mu}
     gave a reasonable range of envelopes: {0.001,5,3000},
     {0.0001,50,30000}, {0.00001,500,300000}, {0.000001,5000,3000000}.
     The time steps are selected in tandem with the scale selection because
     the the code is only semi-implicit and too large of a time step will
     cause an instability.  The scale selects in effect how fast we evolve
     and so smaller scales can be advenced with bigger time steps to get
```

```
     comparable behavior.
```

tik_f.m

```
[tik] = tik_f(fin,lambda)

synopsis:

 This routine calculates the Tikonov regularized version of the
 one dimensional input signal fin.

input:

   fin =  The one dimensional signal to be regularized.
          Assumed to be a row vector.
 lambda = Regularization parameter. The smaller this parameter is,
          the smoother the regularized signal.

output:

   tik = Tikhonov regularized signal.
 It's a row vector of same size as the input fin.

notes:

  1) we found that, given the threaded assembly signals in ticks,
     the lambda values of 5000, 50000, 500000, 5000000 gave
     a nice range of scales to generate Tikhonov regularizions
     with.
```

expert_metric_comp.m

```
[rank,warp] = expert_metric_comp(M,expert,idx)

synopsis:

  compare an expert rating of the comparisons
  with one of the distance (or with minor
  modifcations combinations of the same).
  The results are reported with two scalars:
  rank and warp.

inputs:


   M  =   a cell array of results for
          the comparisons between experimental
          and simulation metric output.
```

```
          Each cell M{i,j}() is assumed to be a
          49 x k matrix, each column of which contains
          the differences of measures
          between the experiment and simulation or
          the various distances.  To be precise, the
          first 36 elements are differences of measures
          -- simulation minus experiment -- the next
          12 are distances, two sets of fractional
           symmetric differences and one set of warping
          distances. The number k is controlled by the
          number of experiments  and equals the number
          of columns in Ce{i,j}().


expert = the expert rankings. It is assumed a 31 x 3
         cell array whose {i,j} element is
         a 1 by size(Ce{i,j},2) row vector
         containing the rankings on a 0-10
         scale: 10 = horrible, terrible,
         deplorable, unspeakable, atrocious
         and 0 is perfect, heavenly, exquisite,
         amazing, (and too good to be true).

      !!! Note: that the expert ratings are reversed!!!
                from the usual high=good low=bad
                system.

  idx =  a selection cell matrix with 1's where the
         {i,j} cell is to be used in the comparison
         and 0's everywhere else.  In test1116 it is
         generated with rand (see test1116 for details)


outputs:


 rank = a number >= 1 which measures the quality of the
        match between expert and metric (the metric is
        selected by Dist below). In particular this
        measures how well the ranking matches between
        the two.

 warp = a measure of how good the match is after the
                ranking comparisons has been done.


notes:

       1) If handed a cell array with expert ranking
          -- say a 31x3 cell array whose {i,j}th cell
          is a 1 by size(Ce{i,j},2) row vector it is
          a simple matter to use M to get a comparison

       2) Dist chooses which of the 49 measures or
          distances to try when comparing with the
          expert scoring.
```

```
details of rank computation:

 If an expert rating has been supplied, compute the comparison
 between the metric selected by Dist (in user parameters).
 We use the following method:

         1) sort expert ratings
         2) use the mapping from the export sort to map
            the metric values.
         3) compute the TV of the thus, quasi-sorted metric
            values.  Divide this by the TV of the fully
            sorted metric values (this is just the max - min)
         4) report this value as Rank1

   now switch the roles of expert and metric ratings and do 1-4
   again to get Rank2  ... this is steps 5-8

         9) report rank = (Rank1 + rank2)/2 as the comparison
            metric between the expert and metric generated
            ratings.

   the closer to one this is, the better.  A value of 1 tells
   that experts and metric agree insofar as RANKINGS go.  There
   can still be significant differences! (Therefore, the warping
   below ...)



details of warp computation:

 Now we measure the difference between the expert and metric ratings
 that cannot be measured by rankings:

 Example: [1, 1.1, 1.2, 1.3, 1.4, 10] = [1, 9.1, 9.2, 9.3, 9.4, 10]
          insofar as ranking is concerned.

 To do this we warp and use the warp energy as a second component
 of the difference between expert and metric

 What is left is the possibility that the top and bottom ranges
 do not match.  We assume they do and begin by mapping the ranges
 to each other so that, remaped, max(s_rM) = max(s_rE) and
 min(s_rM) = max(s_rE).
```